

Learning Selective LLM Autonomy from Copilot Feedback in Enterprise Customer Support Workflows

Anonymous ACL submission

Abstract

We present a deployed system that automates end-to-end customer support workflows inside an enterprise Business Process Management (BPM) platform. The approach is scalable in production and reaches selective automation within two weeks for a new process, leveraging supervision already generated at scale: structured per-case UI interaction traces and low-overhead copilot feedback, where operators either accept a suggestion or provide a correction. A staged deployment pipeline trains a next UI action policy, learns a critic from copilot feedback to calibrate abstention, and executes only high-confidence steps in the background while deferring uncertain decisions to operators and resuming from the updated UI state. This setup lets one operator supervise multiple concurrent sessions and be interrupted only when the system is uncertain. The system operates on a schema-driven view of the BPM interface and includes monitoring and safe fallbacks for production. In production, it automated 45% of sessions and reduced average handling time by 39% without degrading support quality level.

1 Introduction

LLM-based agents promise end-to-end automation by reasoning and acting with tools (Yao et al., 2023; Schick et al., 2023) and show strong results on web and GUI interaction benchmarks (Xie et al., 2024; Drouin et al., 2024). However, deploying such agents in enterprise customer support remains hard: they interleave many model calls with UI actions, so small errors propagate over long horizons; multi-step control loops are costly at scale, and frequent UI and process changes require conservative uncertainty handling to avoid quality regressions. Traditional automation avoids some of these risks but is expensive to build and maintain, relying on manually authored dialogue flows (Qin et al., 2023) or brittle Robotic Process Automation (RPA) scripts (Hindel et al., 2020).

We present a deployed alternative for an enterprise BPM system: instead of hand-authoring automation, we learn workflows directly from operator behavior, reaching selective automation within two weeks without manual rule authoring. Our setting provides two scalable supervision signals: each case yields structured UI interaction traces, and copilot mode provides lightweight feedback: operators either accept a suggestion or correct it.

Using a staged deployment procedure inspired by imitation learning (Ross et al., 2011; Ross and Bagnell, 2014), we progress from offline training on traces to copilot supervision and then to gated background execution, where only high-confidence actions are automated. The system is fully deployed and serves production traffic; we report online A/B results on approximately 15k customers per group, demonstrating a 39% reduction in operator active time without degrading support quality.

Our system combines an LLM policy trained on serialized state-action traces, a critic trained on trajectory feedback labels to support abstention (Geifman and El-Yaniv, 2019; Hendrickx et al., 2024), and a controller that executes only above a calibrated confidence threshold, otherwise deferring to the operator and resuming from the updated UI state. To improve robustness under UI change, the system operates on a schema-driven JSON representation of the BPM interface rather than screenshots.

Our main contribution is a production-scalable staged deployment process that turns BPM UI traces and lightweight copilot feedback into safe, gated background execution. We also report the key engineering components—schema-driven state/action serialization, deployment safeguards (abstention, data drift handling, fallbacks, monitoring), and an online evaluation protocol with ablations—together with measured gains in operator active time.

2 Related Work

Customer support automation has long relied on deployed dialogue and agent-assist systems that suggest replies, retrieve internal knowledge, or summarize conversations for operators (Fadnis et al., 2020; Obadinma et al., 2022; Feigenblat et al., 2021). Newer datasets also capture customer-support interactions, from transactional dialogues to more natural service conversations (Byrne et al., 2021; Gung et al., 2023). Most prior work targets conversation-level assistance or knowledge access, while our setting requires safe execution of multi-step UI procedures inside an enterprise BPM interface.

Early approaches framed UI and workflow control as behavior cloning from interaction logs (Pomerleau, 1988). Over long horizons, covariate shift makes errors compound, which motivates interactive imitation learning methods such as DAgger and AggreVaTe that gather expert supervision on states reached by the learned policy (Ross et al., 2011; Ross and Bagnell, 2014; Sun et al., 2017).

GUI agents also differ in what they observe. Text-based methods act on structured interface signals such as HTML, DOM, or accessibility trees (Gou et al., 2025a; Gur et al., 2024; Lai et al., 2024), while multimodal agents additionally use screen information (Lu et al., 2024; Xu et al., 2025; Tan et al., 2024). In deployed systems, raw DOM is often verbose and fragile under UI changes, which motivates intermediate representations and stronger action grounding (Wu et al., 2024; Gou et al., 2025b). We use the more controlled nature of enterprise BPM to extract a schema-driven JSON state from the DOM that keeps only task-relevant entities and actionable controls, enabling monitoring and reliable tool calling without screenshots. This aligns with industry efforts that restructure support policies and workflows into LLM-friendly formats (Su et al., 2025).

Our deployment loop is DAgger-like. Operator rejections trigger corrective supervision in the same UI state, and we execute selectively with low-confidence handoff to reduce drift over long procedures. Instead of preference-based alignment methods, we use lightweight operator feedback to train a critic and manage the error-coverage trade-off needed for safe background automation (Schulman et al., 2017; Rafailov et al., 2023a; Shao et al., 2024).

3 Framework

Our approach involves a staged deployment algorithm, progressing through three distinct phases: logging, copilot, and selective automation, each building on the data gathered in the previous stage, allowing for gradual integration into the production environment. The deployment is designed to collect data continuously at each stage, ensuring that each transition is based on robust operational feedback.

Problem Setting Operators resolve customer issues via a BPM system integrating full-history chat, scenario search, a workflow workspace for guided UI interactions (e.g., credit requests, account checks, data updates), and auxiliary tabs showing customer products and statuses. Multiple scenarios can run concurrently, with operators interleaving structured UI actions and free-form chat, relying on both encoded business rules and tacit knowledge.

A support session is a trajectory of states s_t and actions a_t : $\{s_t, a_t\}_{t=1}^T$, where s_t comprises chat, UI, scenario, and auxiliary context, and a_t spans UI navigation and interaction, tool invocation, chat messages, or session closure. The automation objective is selective: maximize automated step coverage while deferring uncertain actions to humans to maintain correctness and customer experience.

Data Operator sessions are fully logged, capturing UI state, events such as clicks, edits, submissions, and procedure openings, chat streams, and metadata including timestamps, business line, and routing details. These traces are serialized into structured state-action pairs for behavior cloning. State serialization produces JSON snapshots of the last 30 chat messages, relevant UI context such as customer profile, active scenario, and global announcements, along with operator actions. Actions fall into four categories — message composition, navigation, selection and form-filling, and session management — each targeting a specific interface element identified by its unique `control_id` (see Table 4 in Appendix A for the full action taxonomy). We then allow sufficient time, typically a week or at least one business cycle, to gather comprehensive data reflecting operational patterns.

For LLM input, UI snapshots and actions are merged into a chronological dialog interleaving customer messages, operator actions, and updated UI states. The model predicts the next action, treat-

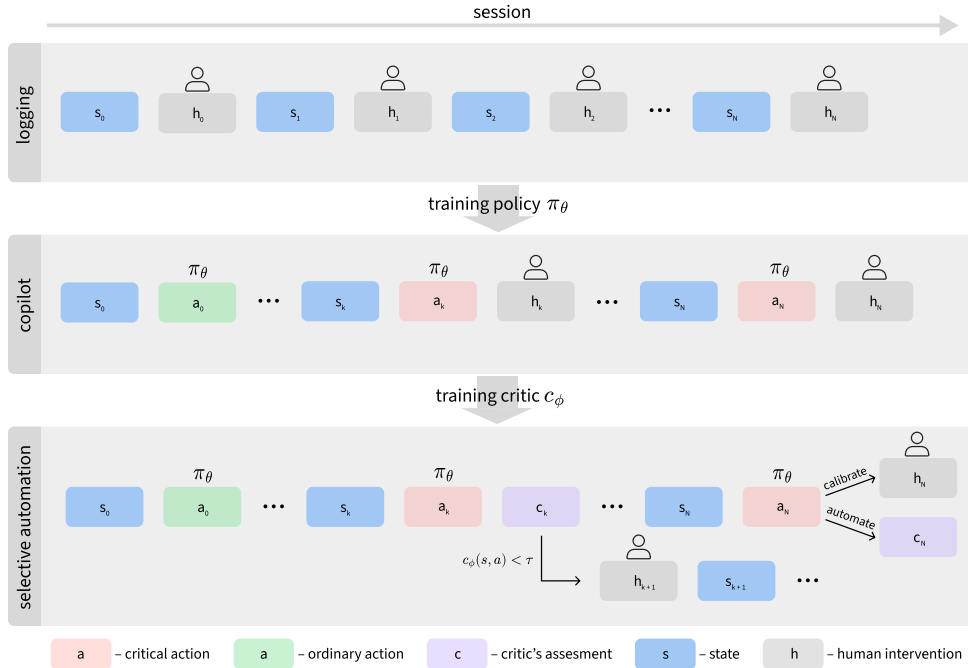


Figure 1: **Staged deployment for selective automation.** *Logging* collects operator trajectories with UI context to train an initial policy. *Copilot* deploys the policy to propose actions with human oversight on critical steps, producing accept or override feedback. *Selective automation* adds a critic that scores policy proposals and executes only critic-approved critical actions. This stage runs in calibration with mandatory human review at the finalizing step and transitions to automation where sessions can be executed end-to-end under calibrated threshold(s) τ .

ing snapshots purely as contextual evidence. Figure 3 summarizes our end-to-end data processing pipeline further details are provided in Appendix A.

Copilot stage Once the logging stage is complete, we proceed to create our first policy, $\pi_\theta(a | s)$, which predicts the next action a based on a serialized state s . The policy is trained in two steps: (1) optional domain-adaptive pretraining on internal texts like knowledge bases, UI descriptions, and chatbot logs, which may boost model performance if available, and (2) direct supervised fine-tuning (SFT) on logs, requiring careful balancing and regularization. The trained models are evaluated using offline metrics, primarily action accuracy, measuring how well predicted actions match actual ones, including their arguments. Once trained, the policy is deployed in copilot mode, where the model proposes actions, and operators either accept or manually correct the suggestion, producing labeled examples of actions. If rejected, the correct markup for the step is obtained for future model iterations. In copilot mode, the system’s performance is monitored through the acceptance rate, the fraction of actions accepted by operators, which is crucial in addressing the multiple paths problem. This allows us to gather human feedback on policy predictions

and assess model quality. Most operator actions involve non-critical steps (like actions that do not change the BPM user state) that do not affect the client or organization, and critical actions (sending messages or terminating interaction) that do. To speed up the process, non-critical actions can be performed automatically by the model, requiring human approval only for critical ones. If an operator disapproves a critical action, they can interact with the interface until the next critical action, after which control returns to the model. This approach results in copilot logs containing accept/reject feedback only for critical actions.

Selective automation stage At this stage, we train a critic $c_\phi(s, a) \in [0, 1]$ from copilot logs to estimate whether a *critical* action a is correct in state s (see Appendix C for details on the critic training). We split selective automation into two substages: *calibration* and *automation*. In calibration, the system executes critic-approved actions during the session, but the finalizing action is always reviewed by a human. The operator either approves finalization or rejects it and performs corrective steps to fix earlier mistakes or complete missed work before closing. In automation, critic-approved critical actions are executed end-to-end,

234 while finalizing actions can remain human-gated
235 until termination reliability targets are met.

236 Execution is regulated by threshold(s) τ , option-
237 ally stratified by action importance. The policy pro-
238 poses $\hat{a} \sim \pi_{\theta}(\cdot | s)$, the critic computes $c_{\phi}(s, \hat{a})$,
239 and the action is executed only when $c_{\phi}(s, \hat{a}) \geq \tau$;
240 otherwise control is handed to a human. The critic
241 is evaluated only for critical actions, while non-
242 critical ones may be executed without this gating.

243 We calibrate τ to meet business precision re-
244 quirements on critical actions. Among critic-
245 approved actions, the fraction that operators would
246 accept must exceed a target, for example 0.9, es-
247 timated from copilot feedback. We initialize τ
248 offline on held-out copilot data, then refine it on-
249 line in the calibration substage without sweeping
250 many thresholds. Online calibration uses a rolling
251 estimate of critical-action precision together with
252 session-level safety signals from calibration, such
253 as the finalization rejection rate and the rate of cor-
254 rective intervention.

255 **Adaptability and Safety** Production deployment
256 must handle out-of-distribution inputs such as new
257 UI elements, rare actions, and unexpected requests.
258 We combine three mechanisms: daily schema
259 and action-distribution validation with automated
260 alerts, critic-based uncertainty gating with stratified
261 thresholds for high-risk actions, and deterministic
262 fallbacks (handoff-to-human, loop detection) when
263 confidence is insufficient. Drift is monitored via
264 persistent calibration traffic that enables periodic
265 re-calibration of τ . Training data is anonymized
266 through entity masking, session-level distinctiv-
267 ness, and selective field removal (details in Ap-
268 pendix B).

269 4 Experiments

270 4.1 Offline Experiments

271 **Experimental Setup** All experiments were con-
272 ducted with the Qwen2.5-Instruct-7B (Team, 2024)
273 model on a single 8xH100 node. We use the 7B
274 size due to production inference constraints: it
275 provides the best trade-off between latency and
276 throughput under our serving budget, while keep-
277 ing GPU memory usage within limits compared
278 to larger checkpoints. The full set of training hy-
279 perparameters is reported in Table 5. We eval-
280 uate two models offline on historical interaction
281 logs: an action model and a critic model. The ac-
282 tion model is assessed on tool and argument selec-
283 tion using strict string matching for most actions,

284 and a Levenshtein-based fuzzy matcher for text-
285 generation actions, with normalized scores thresh-
286 olded to tolerate minor paraphrases. The evaluation
287 set is drawn from real user interactions across two
288 sources: copilot mode and fully automated mode.
289 We report results separately, as differences in oper-
290 ator behavior, action timing, and available context
291 introduce a mild distribution shift.

292 **Metrics** For the action model, we report tool-
293 and action-level accuracy (overall and stratified by
294 action type). The critic model is a binary classifier;
295 we therefore report precision, recall, and F1.

296 **Baseline** As a baseline, we use the same agentic
297 prompting setup but without state-action SFT and
298 without domain pretraining.

299 **Do we need SFT policy training?** We evalu-
300 ate several zero-shot prompted models — includ-
301 ing leading open-source and proprietary ones —
302 against our 7B SFT policy, all receiving identical
303 session context (prompt in Appendix G). As shown
304 in Table 1, even the strongest prompted model
305 reaches only 47.78% tool accuracy compared to
306 79.15% for SFT, and the gap persists across all
307 model families and scales. The comparison does
308 not claim general model superiority; it establishes
309 that the task domain is sufficiently specialized that
310 domain-specific SFT is a prerequisite for reliable
311 BPM automation.

Model	Tool Acc.	Action Acc.
Qwen2.5-Instruct-7B (baseline)	34.95	10.27
DeepSeek V3.1 (deepseek-ai, 2025)	42.40	27.83
GPT-5.2 (OpenAI, 2025)	43.01	27.02
Gemini 3 Pro (Google DeepMind, 2025)	47.78	32.93
Qwen3-235B-A22B (Yang et al., 2025)	44.33	26.18
Ours	79.15	65.48

Table 1: Action accuracy of zero-shot prompting across model scales vs. domain-specific SFT, demonstrating that task-specific fine-tuning is necessary regardless of model capacity.

312 **How should historical production logs be mixed
313 with human rejections?** We examine whether
314 adding operator corrections from copilot deploy-
315 ment improves beyond SFT on historical logs alone.
316 Rejected samples — where operators override pol-
317 icy suggestions and log corrective actions — are
318 collected over one week of production copilot us-
319 age and are mixed with historical logs at varying
320 ratios. A 2.5k-sample copilot-rejection test set eval-
321 uates generalization to hard cases. As shown in

Table 2, adding rejected correction samples boosts accuracy on the copilot rejection set, and the best trade off is achieved by combining 170k predefined log samples with 40k rejected samples while keeping predefined accuracy nearly unchanged. We did

$N_{\text{predefined}}$	N_{rejected}	$\text{Acc}_{\text{predefined}}$	$\text{Acc}_{\text{rejected}}$
170k	0	71.25	14.88
100k	10k	59.10	17.60
100k	20k	69.77	21.76
100k	40k	69.52	25.74
40k	40k	59.11	19.96
50k	20k	67.25	22.92
170k	40k	71.07	25.72

Table 2: Effect of mixing historical logs with copilot rejected corrections on predefined and rejected accuracy.

not observe gains from single-step preference optimization on the same feedback, likely due to tool dataset imbalance and noisy rejections; results are in Appendix C.

Why do operators reject copilot suggestions?

We analyzed 9370 rejected actions from a 4-day period in production copilot mode to build an error taxonomy. Rejections were sorted into broad groups by action type, and human annotators reviewed 100 samples per group to find recurring patterns (see Appendix E for the annotation protocol and detailed category descriptions). These patterns formed a taxonomy, and proportions were estimated for all rejections. The results revealed four main rejection reasons, as shown in Figure 2. Notably, 37.2% of rejections came from operator preferences or optional steps, meaning the accept/reject signal is noisy and suggesting the need for more detailed feedback like graded labels. These findings highlight the importance of separating operator preferences from incorrect suggestions, better handling of environment limitations, targeted improvements for model errors in routing and context handling, as well as stronger operator onboarding around copilot usage to improve critic training and automation.

Additional Experiments. We conduct ablation studies to inform key design choices in Appendix C. Scaling SFT data consistently improves accuracy, but balancing by screen or tool name offers no benefit over natural-distribution — uniform tool balancing actually hurts performance, suggesting real operator frequency patterns carry useful signal (Table 6). Finally, an SFT-trained critic using human preference feedback outperforms a confidence-

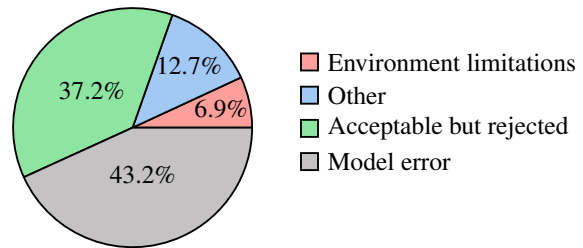


Figure 2: High-level breakdown of rejected suggestions.

based baseline by +3.84% precision and +3.03% F1 (Table 7), validating explicit supervision over raw model uncertainty for action gating.

4.2 Online Experiments

Experimental Setup After training on copilot logs, we rolled out the policy-critic pipeline to production. We then assessed its real-world behavior on live traffic and quantified the end-to-end impact via two controlled A/B tests against production baselines. More details on A/B test groups are discussed in Appendix D.

Metrics We report **Acceptance rate** for critical suggestions in copilot mode; **Automation rate**, defined as the fraction of sessions completed end-to-end without operator intervention (sessions that only wait for customer reply are counted as automated if no reply is received within a timeout, with subsequent messages treated as a new session); **Resolution correctness**, measured via blind operator evaluation on a 0–100 rubric (Appendix F); and **AAT**, the average agent (operator) active time per customer, where fully automated sessions contribute zero.

Baselines We compare the copilot pipeline against two production baselines: a chatbot-assisted workflow with curated intents and scripted UI steps, and the standard human-operated workflow in the BPM without copilot or automation.

How well are policy suggestions accepted? We measure the acceptance rate of critical policy suggestions in production, averaged over one week. Operators accept deterministic UI actions at a high rate: `click_control` (86.73%), `close_chat` (84.81%), and `transfer_chat` (91.7%) — indicating strong agreement with the policy’s proposed operational steps. In contrast, `send_text_to_chat` is accepted less frequently (50.81%), which is expected because free-form responses are more

Exp	Control→Treatment	N/grp	Days	AAT↓	ΔAAT	p	CI
E1	Human-only→Selective (auto)	15.5k	7	227.37→139.15	-39%	<0.05	[-42% , -35%]
E2	Classic bot→Selective (auto)	77k	32	124.67→104.72	-16%	<0.05	[-20% , -12%]
E3	Selective (confirm)→Selective (auto)	17k	7	187.66→150.13	-25%	<0.05	[-28% , -23%]

Table 3: Online customer-level A/B tests. E1 compares a human-only BPM workflow to selective automation with automation-enabled execution. E2 compares a classic bot baseline to our copilot pipeline with automation-enabled execution. E3 compares selective automation with mandatory final confirmation to the same selective setup with automation-enabled execution. CI denotes the 95% confidence interval for the Δ AAT effect.

sensitive to tone, wording, and policy constraints. Overall, the results suggest that procedural actions are already reliable in production, while natural-language messaging remains the main bottleneck for higher automation.

Does it reduce operator time? Compared with a human-only baseline, automation-enabled selective operation reduces AAT from 227.37 s to 139.15 s, a 39% decrease that is statistically significant at $p < 0.05$, as shown in E1, Table 3. Resolution correctness remains on par with the baseline, showing no statistically significant difference. In automation-mode, the system fully automates 45% of sessions.

Does it outperform a classic bot baseline? Relative to a production baseline that combines a chatbot with scripted steps, our copilot pipeline reduces AAT from 124.67 s to 104.72 s, a 16% decrease that is statistically significant at $p < 0.05$, as reported in E2, Table 3. Resolution correctness remains unchanged. On the operational side, new procedures are rolled out on a roughly two-week deployment cycle, and ongoing effort consists primarily of monitoring and periodic continual improvement rather than the continuous scripting and brittle workflow maintenance typical of RPA systems. Further details are provided in Appendix B.

Does end-to-end automation beat copilot-style operation? Within the selective workflow, moving from mandatory final confirmation to automation-enabled execution reduces AAT from 187.66 s to 150.13 s, a 25% decrease that is statistically significant at $p < 0.05$, as shown in E3, Table 3. Resolution correctness remains statistically unchanged. This result isolates the incremental benefit of removing the confirmation bottleneck: when execution can proceed automatically, operator time shifts from repeated action-level handling toward issue diagnosis and closure validation, yielding additional throughput gains.

5 Conclusion

We presented a deployed system that progressively moves from copilot supervision to selective automation inside an enterprise BPM platform. Instead of full autonomy, we treat automation as incremental rollout: collect lightweight accept/override feedback from operators, then gate execution so only high-confidence steps run automatically while uncertain cases defer to humans. In production, the largest gains come from automating high-frequency, low-risk procedural work while preserving safety through abstention. The approach is more adaptable than rule-based automation and more predictable than agentic control loops, leveraging naturally occurring supervision and supporting continuous adaptation via monitoring and retraining. The recipe generalizes to any repeated-action workflow in a structured interface where interaction logs are available.

Limitations

Our results come from proprietary data within a single enterprise ecosystem, limiting reproducibility and cross-domain generalization. Automation is deliberately restricted to low-risk actions with operator approval on higher-risk steps, so reported gains do not reflect unconstrained autonomy. Expanding reliable coverage to long-horizon and language-sensitive steps without sacrificing safety remains underexplored. The approach depends on critic thresholds calibrated from offline proxies with limited online refinement, implicitly assuming these signals stay correlated with business metrics under shifting traffic and workflows. As automation reaches rarer scenarios with sparse supervision, both policy and critic can degrade in ways that are hard to detect in aggregated metrics, motivating conservative gating, slice-based monitoring, and rapid rollback capabilities.

Ethical Statement

Our work deploys an LLM-based automation system for customer-support workflows. The data may include sensitive information, so all training and evaluation were conducted inside a controlled enterprise environment with strict access control. We reduce privacy risk by masking direct identifiers when possible, limiting logs to what is necessary for operation and evaluation, and applying retention and auditing practices aligned with internal security policies. All examples in this paper are synthetic or carefully redacted.

To reduce harm from incorrect automation, the system uses selective automation with action-level risk tiering. Low-risk actions, such as navigation and routine interface steps, can be executed with minimal friction. High-risk actions that may affect the customer, such as sending messages or making irreversible account changes, require explicit human approval. A learned critic can abstain and defer to an operator, or fall back to a safer baseline when confidence is low or inputs are out of distribution. Operators can review, accept, or reject proposed actions, and we monitor safety and quality signals such as deferrals, rollbacks, and task outcomes. We also check performance across heterogeneous issue types and other relevant slices, and tighten automation thresholds when risks or disparities appear.

Operator impact. The system is designed to scale support capacity to a growing customer base rather than to reduce headcount. In practice, automation removes repetitive procedural work and shifts operator focus toward complex cases requiring judgment. To support this transition, operators undergo onboarding that frames their evolving role as AI editors who review and refine system outputs. Qualitative feedback from operators has been positive, with operators reporting reduced burden from routine tasks. Reproducibility is limited by proprietary data, so we report aggregated results and sufficient methodological details to support assessment.

References

Bill Byrne, Karthik Krishnamoorthi, Saravanan Ganesh, and Mihir Kale. 2021. *TicketTalk: Toward human-level performance with end-to-end, transaction-based dialog systems*. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference*

on Natural Language Processing (Volume 1: Long Papers), pages 671–680. Association for Computational Linguistics.

deepseek-ai. 2025. *Deepseek-v3.1 (model card)*. Accessed 2026-02-14.

Karel D’Oosterlinck, Winnie Xu, Chris Develder, Thomas Demeester, Amanpreet Singh, Christopher Potts, Douwe Kiela, and Shikib Mehri. 2025. *Anchored preference optimization and contrastive revisions: Addressing underspecification in alignment*. *Transactions of the Association for Computational Linguistics*, 13:442–460.

Alexandre Drouin, Mathieu Bastien, Ishan Jindal, and 1 others. 2024. *Workarena: How capable are web agents at solving common knowledge work tasks?* *arXiv preprint arXiv:2403.07718*.

Kshitij Fadnis, Winnie N Mohan, Jatin Tinganhotra, Luis Lastras, Haggai Roitman, Doron Cohen, Yosi Mass, Shaie Haie, and David Konopnicki. 2020. *Agent assist through conversation analysis*. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 151–157. Association for Computational Linguistics.

Guy Feigenblat, Chulaka Gunasekara, Benjamin Sznajder, Sachindra Joshi, David Konopnicki, and Ranit Aharonov. 2021. *TWEETSUMM - a dialog summarization dataset for customer service*. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 245–260. Association for Computational Linguistics.

Yonatan Geifman and Ran El-Yaniv. 2019. *Selectivenet: A deep neural network with an integrated reject option*. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2151–2159. PMLR.

Google DeepMind. 2025. *Gemini 3 pro model card*. Model card. Accessed 2026-02-14.

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jiménez Gutiérrez, Yiheng Shu, Chan Hee Song, Jiaman Wu, Shijie Chen, Hanane Nour Moussa, Tianshu Zhang, Jian Xie, Yifei Li, Tianci Xue, Zeyi Liao, and 7 others. 2025a. *Mind2web 2: Evaluating agentic search with agent-as-a-judge*. *CoRR*, abs/2506.21506.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025b. *Navigating the digital world as humans do: Universal visual grounding for gui agents*. *Preprint*, arXiv:2410.05243.

James Gung, Emily Moeng, Wesley Rose, Arshit Gupta, Yi Zhang, and Saab Mansour. 2023. *NatCS: Eliciting natural customer support dialogues*. In *Findings of*

583	<i>the Association for Computational Linguistics: ACL 2023</i> , pages 9652–9677. Association for Computational Linguistics.	Stéphane Ross and J. Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning . <i>CoRR</i> , abs/1406.5979.	637
584			638
585			639
586	Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis . <i>Preprint</i> , arXiv:2307.12856.	Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning . In <i>Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011</i> , volume 15 of <i>JMLR Proceedings</i> , pages 627–635. JMLR.org.	640
587			641
588			642
589			643
590			644
591	Kilian Hendrickx, Lorenzo Perini, Dries Van der Plas, Wannes Meert, and Jesse Davis. 2024. Machine learning with a reject option: A survey . <i>Machine Learning</i> .		645
592			646
593			647
594			
595	Julia Hindel, Lena Cabrera, and Matthias Stierle. 2020. Robotic process automation: Hype or hope? In <i>WI2020 Centrale Tracks</i> , pages 1750–1762.	Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools . In <i>Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023</i> .	648
596			649
597			650
598	Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. Autowebglm: A large language model-based web navigating agent . <i>Preprint</i> , arXiv:2404.03648.		651
599			652
600			653
601			654
602		John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms . <i>CoRR</i> , abs/1707.06347.	655
603	Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. Omniparser for pure vision based gui agent . <i>Preprint</i> , arXiv:2408.00203.		656
604			657
605			658
606	Stephen Obadinma, Faiza Khan Khattak, Shirley Wang, Tania Sidhom, Elaine Lau, Sean Robertson, Jingcheng Niu, Winnie Au, Alif Munim, Karthik Raja K. Bhaskar, and 1 others. 2022. Bringing the state-of-the-art to customers: A neural agent assistant framework for customer service support . In <i>Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track</i> , pages 440–450. Association for Computational Linguistics.	Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models . <i>CoRR</i> , abs/2402.03300.	659
607			660
608			661
609			662
610			663
611			664
612			665
613			666
614			667
615	OpenAI. 2025. Update to gpt-5 system card: Gpt-5.2 . System Card. Accessed 2026-02-14.	Hanchen Su, Wei Luo, Yashar Mehdad, Wei Han, Elaine Liu, Wayne Zhang, Mia Zhao, and Joy Zhang. 2025. LLM-friendly knowledge representation for customer support . In <i>Proceedings of the 31st International Conference on Computational Linguistics: Industry Track</i> , pages 496–504. Association for Computational Linguistics.	668
616			669
617	Dean Pomerleau. 1988. ALVINN: an autonomous land vehicle in a neural network . In <i>Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]</i> , pages 305–313. Morgan Kaufmann.	Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. 2017. Deeply aggravated: Differentiable imitation learning for sequential prediction . In <i>Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017</i> , volume 70 of <i>Proceedings of Machine Learning Research</i> , pages 3309–3318. PMLR.	670
618			671
619			672
620			673
621			674
622	Libo Qin, Wenbo Pan, Qiguang Chen, Lizi Liao, Zhou Yu, Yue Zhang, Wanxiang Che, and Min Li. 2023. End-to-end task-oriented dialogue: A survey of tasks, methods, and future directions . <i>arXiv preprint arXiv:2311.09008</i> .	Weihaio Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, Ruyi An, Molei Qin, Chuqiao Zong, Longtao Zheng, Yujie Wu, Xiqiang Chai, Yifei Bi, Tianbao Xie, Pengjie Gu, and 9 others. 2024. Cradle: Empowering foundation agents towards general computer control . <i>Preprint</i> , arXiv:2403.03186.	675
623			676
624			677
625			678
626			679
627	Rafael Rafailov, Kevin Lee, Jimmy Ba, and Michael Zhao. 2023a. Direct preference optimization: Your language model is secretly a reward model . <i>arXiv preprint arXiv:2305.18290</i> .		680
628			681
629			682
630			683
631	Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023b. Direct preference optimization: Your language model is secretly a reward model . In <i>Advances in Neural Information Processing Systems</i> , volume 36.	Qwen Team. 2024. Qwen2.5: A party of foundation models .	684
632			685
633			686
634			687
635			688
636		Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2024. Os-atlas: A foundation action model for generalist gui agents . <i>Preprint</i> , arXiv:2410.23218.	689

694 Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan
695 Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhou-
696 jun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu,
697 Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming
698 Xiong, Victor Zhong, and Tao Yu. 2024. [Osworld:
699 Benchmarking multimodal agents for open-ended
700 tasks in real computer environments](#). In *Advances in
701 Neural Information Processing Systems 38: Annual
702 Conference on Neural Information Processing Sys-
703 tems 2024, NeurIPS 2024, Vancouver, BC, Canada,
704 December 10 - 15, 2024*.

705 Hai-Ming Xu, Qi Chen, Lei Wang, and Lingqiao Liu.
706 2025. [Attention-driven gui grounding: Leveraging
707 pretrained multimodal large language models without
708 fine-tuning](#). *Proceedings of the AAAI Conference on
709 Artificial Intelligence*, 39(8):8851–8859.

710 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang,
711 Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao,
712 Chengen Huang, Chenxu Lv, Chujie Zheng, Day-
713 iheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao
714 Ge, Haoran Wei, Huan Lin, Jialong Tang, and 40
715 others. 2025. [Qwen3 technical report](#). *CoRR*,
716 abs/2505.09388.

717 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak
718 Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023.
719 [React: Synergizing reasoning and acting in language
720 models](#). In *The Eleventh International Conference
721 on Learning Representations, ICLR 2023, Kigali,
722 Rwanda, May 1-5, 2023*. OpenReview.net.

A Data Pipeline

Interaction Logging We set up the BPM system to record full operator sessions. Each log entry includes the current UI state visible to the operator, covering both customer-facing and back-office information, as well as UI events like button clicks, form edits and submissions, table selections, and procedure openings. It also captures the chat history, meaning messages from the customer and replies from the operator, along with extra metadata such as timestamps in both system and customer time zones, business line, and routing details. These raw interaction logs serve as the basis for our pipeline: they are later converted into state–action pairs and used to train our policy through behavior cloning.

State Serialization and Action Space A key part of our data pipeline is state serialization. We build a structured JSON snapshot of the operator’s interface and the ongoing conversation using manually designed rule-based parsers. Unlike web-browsing agents that deal with unconstrained and highly variable HTML, agents in a BPM environment work with a limited and well-defined set of interface states. We take advantage of this property to reduce noise and extract only the elements needed to resolve the customer’s request.

More specifically, we serialize the last 30 chat messages, updating the window when the operator scrolls, along with the UI information used during issue resolution, such as the customer profile, the active scenario or procedure, and global context like incident announcements. We also capture the operator’s actions performed in the interface.

We distinguish four categories of actions, as listed in Table 4: composing and sending messages to the customer, navigation actions that update the UI state, such as switching screens or opening procedures; selection and form-filling actions that modify fields on the current screen, and session-management actions that close the current interaction or transfer the chat to another operator. Each recorded UI action targets a specific interface element identified by its unique `control_id`.

Converting Serialized States to LLM Input

The final step is to combine all serialized features from an interaction session into a single model-ready dialog in a chat format. We build the dialog by interleaving customer and operator messages with UI interactions in chronological order, mean-

Category	Action
Message Composition	send_text_to_chat
Navigation	open_procedure click_control
Selection & Form-filling	select_radio_button select_element_in_combo_box select_element_in_select fill_input
Session Management	close_chat transfer_chat

Table 4: Action Categories and Commands

ing we represent textual messages, UI snapshots, and actions within a single session timeline. At the beginning of the session, we place the initial UI snapshot together with the chat messages observed up to that moment. Then we iteratively add each operator action followed by the corresponding updated UI snapshot.

Both actions and snapshots are encoded as textual messages. During training, the model is trained to predict the next action, while UI snapshots serve only as contextual evidence and are not treated as prediction targets.

B Adaptability and Safety

OOD Handling One of the biggest challenges in production is dealing with situations that fall outside the distribution of the training logs. Examples of such situations are new UI elements or changes in the interface layout, rare action sequences or corner cases that were never logged before, and unexpected client requests or completely new process flows.

To handle these out-of-distribution (OOD) scenarios, we put several safeguards in place:

- **Daily data and log validation:** New or unknown UI elements are caught by automatic validation rules. Every day we calculate the share of sessions that fail validation and record a breakdown of failure rates for each rule. If error ratios go above the thresholds, which are set based on acceptable frontend or backend update delays, an alert is sent out for further investigation.
- **Critic-based uncertainty detection:** Whenever the critic score $c_\phi(s, a)$ drops below a threshold τ , the action is forwarded to a human operator instead of being executed. This protects clients from hasty or poorly justified

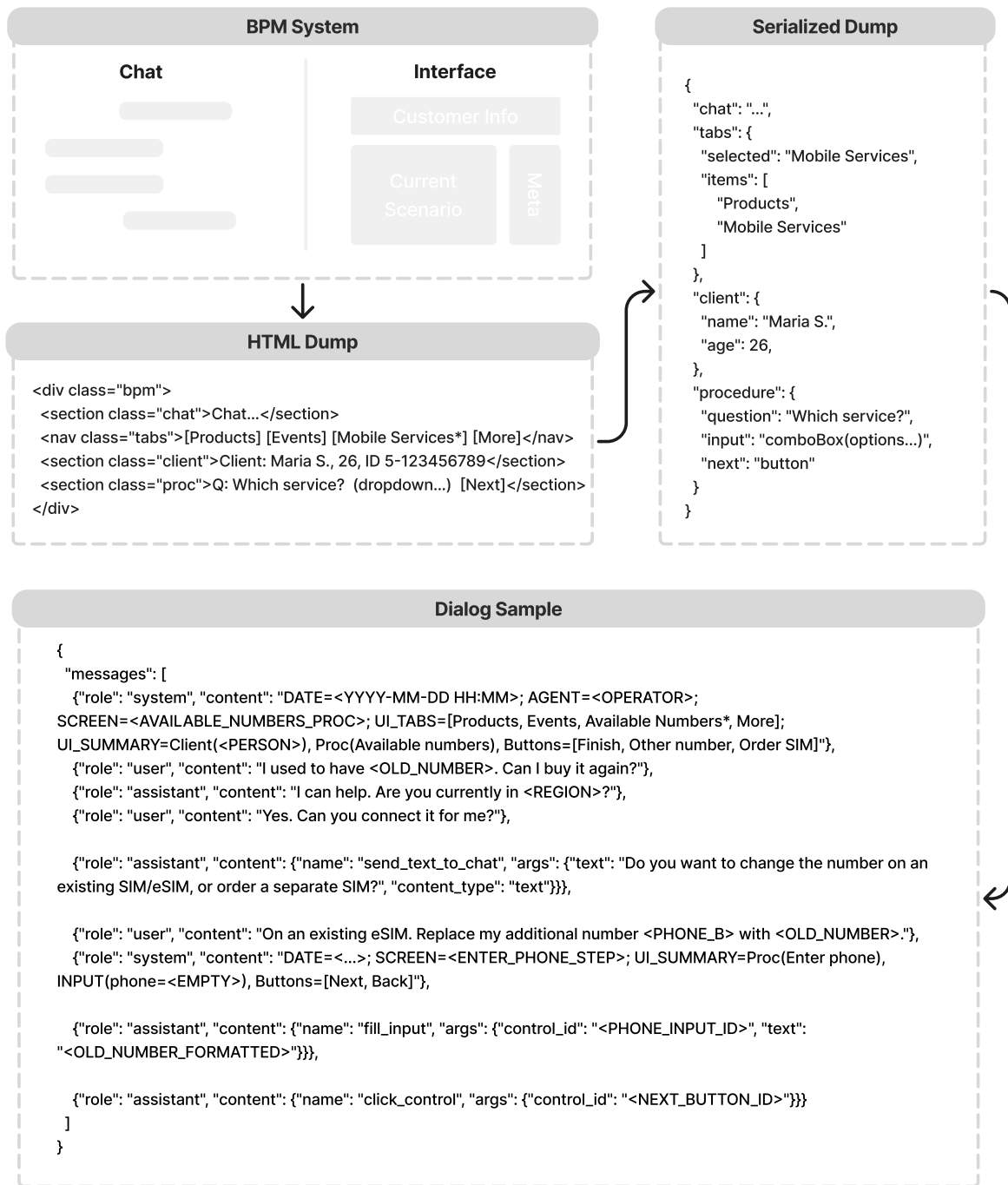


Figure 3: Data pipeline. We first log each BPM step as an HTML snapshot that includes both the customer chat and the visible UI state. We then convert the raw HTML into a task-centric JSON representation using a set of hand-written parsers. Finally, we merge the JSON state with the dialog history to produce model-ready training/inference samples.

810 model actions in real time. It also helps us find
811 model weaknesses later, by analyzing logs of
812 low-confidence or rejected actions.

813 • **Fallback policies:** For cases that are still not
814 covered, we use deterministic or rule-based
815 fallback procedures to guarantee safety and
816 compliance. These procedures also step in to
817 stop execution chains that have clearly gone
818 off track — for instance, when a loop is de-
819 tected or the sequence of actions no longer
820 matches the expected process logic.

821 Together, these mechanisms let the system work
822 reliably even in new or unusual situations, while
823 keeping safety and support quality high.

824 **Continuous Improvement Loop** The method
825 described above forms the core of our approach,
826 but to meet the demands of production we also
827 need to make sure it stays flexible. In practice, the
828 topics of customer requests keep changing, and
829 operator scripts are updated regularly. Because
830 of this, we need a fast and reliable way to test
831 new ideas about model input, architecture, or small
832 pipeline features, and to validate them on up-to-
833 date data.

834 To meet these needs, we maintain a continuous
835 training and validation loop built around the fol-
836 lowing components:

837 • **Dynamic datasets.** Our datasets are not kept
838 fixed for a long time. Each dataset covers one
839 business cycle or one series of experiments
840 and is replaced afterwards. Since our process
841 already provides ground-truth actions and ac-
842 cept/reject labels from human operators, we
843 only need a reusable set of data preparation
844 scripts and make sure they stay up to date.

845 • **Slice-aware operation.** Since production traf-
846 fic is heterogeneous, we segment sessions
847 into issue clusters (e.g., by topic) and train
848 a dedicated policy–critic pair per cluster. This
849 makes drift localized: regressions typically
850 appear in a small slice and can be diagnosed
851 and fixed without globally tightening automa-
852 tion. A practical constraint is the long tail of
853 small clusters with limited gold demonstra-
854 tions; in such slices critic-based abstention
855 may be unreliable, so we keep them in copilot-
856 only mode (no selective automation) until suf-
857 ficient supervision accumulates.

858 • **Online metrics monitoring.** Besides data
859 quality, the quality of the support itself must
860 also remain high. For this reason, we set up
861 alerts for drops in online metrics and moni-
862 tor them on an ongoing basis. For each clus-
863 ter we track a set of guardrails that combine
864 calibration-based estimates of critical-action
865 precision and session-level safety signals such
866 as finalization rejection and corrective inter-
867 vention rates with business guardrails such as
868 returns and CSAT.

869 • **Slice-level guardrails and rollback.** When
870 any guardrail triggers, we automatically dis-
871 able selective automation for the affected
872 cluster and fall back to copilot-only, routing
873 critical actions to operators to collect fresh
874 supervision. Using these newly collected
875 operator traces, we retrain the policy and
876 critic on a rolling window and rerun the stan-
877 dard selective-automation pipeline, including
878 mandatory recalibration of the critic thresh-
879 old, before redeploying the updated pair in
880 calibration and re-enabling automation.

881 These practices give us high adaptability to process
882 changes and short time-to-market for new improve-
883 ments.

884 **Safety and Compliance Constraints** In a real-
885 world customer support setting, production logs
886 inevitably contain many kinds of personal and sen-
887 sitive information, such as customer names, contact
888 details, account identifiers, and so on.

889 To comply with privacy regulations and internal
890 security policies, all datasets used for model train-
891 ing, validation, and evaluation go through strict
892 anonymization and filtering procedures. Before
893 any data enters the training pipeline, we carry out
894 the following steps:

895 • **Data classification and masking:** Each
896 log record is parsed to detect personal data
897 fields using a combination of rule-based
898 detectors (regular expressions, entity dic-
899 tionaries) and trained named entity recog-
900 nition (NER) models. Detected entities
901 are replaced with neutral placeholders (e.g.,
902 <NAME>, <PHONE_NUMBER>), so that model in-
903 puts keep their structure and meaning but con-
904 tain no personal content.

905 • **Session-level distinctiveness:** Some identi-
906 fiers need to remain distinguishable within

the same session. To achieve this, different personal data instances of the same type are replaced with masks that have different letter suffixes (e.g., <NAME_A>, <PHONE_NUMBER_B>).

- **Selective field removal:** For certain data types, such as attachments, contract documents, or images, the entire field is removed from the dataset to eliminate any chance of leakage.

These safeguards ensure that no personal information is used during model training or evaluation.

C Offline Experiments

Hyperparameters Table 5 summarizes the full set of SFT training hyperparameters.

Parameter	Value
Model	
Base model	Qwen2.5-Instruct-7B
Max sequence length	32768
Precision	bfloat16
Training	
Training duration	2 epochs
Global train batch size	32
Microbatch size (per device)	1
Gradient clipping	norm, threshold 2.0
Optimizer and scheduler	
Optimizer	Decoupled AdamW
Learning rate	$2 \cdot 10^{-6}$
β_1, β_2	(0.9, 0.95)
ϵ	10^{-8}
Weight decay	10^{-7}
LR schedule	cosine with warmup
Warmup	500 batches
Final LR fraction (α_f)	0.1
Evaluation	
Eval interval	every 500 batches
Eval subset	20 batches
Distributed training (FSDP)	
Sharding strategy	FULL_SHARD
State dict type	sharded
Activation checkpointing	enabled
Forward prefetch	enabled
Backward prefetch	BACKWARD_PRE
CPU offload	disabled

Table 5: Hyperparameters for SFT model training.

Sample Size and Balancing Ablation We study how the amount of SFT data and different balancing strategies affect policy performance. We compare three sampling strategies for training sets of size N from 4k to 64k:

- Random sampling from the natural state-action production logs distribution without any balancing.
- Balancing by BPM screen, where we sample uniformly across screen buckets.
- Uniform balancing by the tool name that is referenced in the action.

All models are trained with the same architecture, hyperparameters, and we evaluate them on a held-out set using argument accuracy. The results in Table 6 show that increasing the dataset size consistently improves performance. However, we do not observe a consistent benefit from balancing compared to random sampling. In fact, balancing by tool name performs worst across all sample sizes. This finding suggests that making tool frequencies uniform removes the informative structure of real operator behavior and harms the model’s ability to generalize.

Dataset Size	No Balancing	By Screen	By Tool
4k	55.92	55.80	50.97
8k	61.47	62.58	57.47
16k	67.15	66.82	62.02
32k	69.60	69.22	67.47
64k	71.19	70.77	69.28

Table 6: Argument accuracy on the held-out set for different training set sizes and balancing strategies. Increasing the dataset size consistently improves performance, while balancing by tool name leads to the lowest accuracy across all sizes. Best results per row are highlighted in bold.

Critic Architecture and Training The critic $c_\phi(s, a)$ is initialized from the same Qwen2.5-7B checkpoint as the policy and fine-tuned as a binary classifier on copilot accept/reject labels. The input is the same serialized state–action dialog used by the policy, appended with the proposed action; the model is trained to generate a single token (accept or reject) with the softmax probability of accept serving as the scalar score $c_\phi(s, a) \in [0, 1]$. We train the critic using the same fine-tuning hyperparameters as the policy SFT stage. We found this generative framing more stable than adding a classification head, as it reuses the policy’s learned representations without introducing new parameters. The critic is evaluated only on critical actions at inference time; non-critical actions bypass the gate entirely.

Method	Precision	Recall	F1
Action confidence	70.84	73.43	70.96
SFT	74.68	73.53	73.99

Table 7: Offline evaluation of critic model variants on binary action decision prediction, where the model decides whether to execute or defer a proposed action. The SFT critic yields notable gains in Precision and F1 while maintaining comparable recall.

Comparing uncertainty-based critic with SFT.

We evaluate two approaches for the critic model. The task is framed as a binary classification problem: the model predicts whether a proposed action should be executed or rejected. As a baseline, we use the action model’s overall confidence score as a proxy for executability. In other words, we treat the model’s confidence in the predicted action as the critic signal. Our main approach trains a dedicated critic through SFT. This critic is initialized from the action model and then fine-tuned on historical interaction logs that contain human feedback on suggested actions, specifically whether they were accepted or rejected. We report Precision, Recall, and F1 on an offline test set.

The results in Table 7 show that the critic trained with SFT consistently outperforms the confidence-based baseline. Precision improves by 3.84%, and F1 increases by 3.03%, while recall remains at a comparable level with only a 0.10 % difference. These findings suggest that learning from historical accept and reject feedback produces a more reliable execution signal than relying on the action model’s raw confidence alone.

Offline Single-Step Preference Tuning Learning from operator feedback is important for *selective automation*: incorporating corrections on the cases where the copilot policy fails increases coverage while maintaining safety. In our main approach, we already leverage this feedback via SFT by training on copilot-mode failures, replacing the rejected policy action with the operator-corrected action.

However, human feedback is often used with *preference-based* alignment objectives rather than pure supervised learning. We therefore evaluate offline single-step preference tuning on copilot data by constructing paired examples from operator overrides: for a rejected proposal a^- in state s , we treat the operator correction a^+ as preferred and optimize Direct Preference Optimization (DPO, (Rafailov et al., 2023b)) and Anchored Preference

Optimization (APO, (D’Oosterlinck et al., 2025)) on $(s, a^+ \succ a^-)$ pairs.

Copilot feedback can only be used in a *single-step* setup in our logging regime: when an operator rejects an action, the policy’s action chain is interrupted and we do not observe the continuation of the model’s trajectory, preventing multi-turn preference learning from these logs alone.

Empirically, single-step preference tuning degrades performance on the production distribution (Table 8). A likely cause is *distribution shift*: constructing a paired dataset from rejected events overweights difficult, operator-intervened states and moves the training distribution away from historical production logs, which hurts metrics on the production bucket.

In addition, operator rejections are noisy as a preference signal. Figure 2 shows that a large fraction of rejected actions were still *acceptable*, implying that many $(a^+ \succ a^-)$ pairs encode stylistic or risk-aversion preferences rather than correctness, injecting substantial label noise into the preference dataset.

Finally, copilot rejection pairs introduce a strong *tool imbalance* and repeated “template” pairs (e.g., the operator selecting `click_control` instead of `send_text_to_chat` in similar contexts). This makes it easy for the model to *hack* the objective by systematically shifting probability mass toward a frequent tool, rather than learning the causal decision rules needed for correct action selection.

A promising direction is multi-turn alignment, which would address most of the above issues: operators can more reliably judge full action sequences than isolated steps under path multiplicity (an individual step may be acceptable depending on what the policy does next); and trajectory-level feedback reduces action/tool imbalance since supervision is assigned to an entire rollout rather than a single tool choice. However, moving to multi-turn alignment is technically challenging: it requires an offline environment that enables policy exploration in BPM engine and assigns outcome-based rewards to completed trajectories. We leave this direction for future work.

D Online Experiments

This appendix provides additional information on online experiments and A/B tests that have been performed.

All tests use customer-level randomization, so

Method	Action Acc. (%)
SFT (baseline)	76.03
SFT + APO	68.73 (-7.30)
SFT + DPO	69.97 (-6.06)

Table 8: Effect of single-step preference tuning on action prediction accuracy. Both DPO and APO models are initialized from the SFT baseline.

control and test are exposed to the same seasonality and campaigns, and receive comparable mixes of customer issues. Escalations are handled by the same pool of operators in both groups under identical staffing and SLAs. Although randomization is at the customer level, we verified balance by comparing pre-period (pre-experiment) metrics between groups, including traffic volume, escalation rate, and baseline AAT, and found no material differences.

We compare against two established production baselines. Human-only is the standard workflow where customer chats are handled end-to-end by human operators without copilot or automation. Classic bot is a production rule-based chatbot that routes users into curated intent flows for common issues. When the bot cannot resolve the request—e.g., the user presses the “issue not resolved” button or explicitly asks to talk to an operator—it hands off the chat to a human operator, who continues in the same workflow.

E Error Taxonomy of Rejected Copilot Suggestions

E.1 Annotation protocol.

We start by sorting all rejected events into broad groups based on the type of action the system suggested and what the operator actually did instead. For example, the policy might suggest sending a text message to the chat, but the operator rejects it and performs a completely different action. We use simple rules to create these groups. Table 9 shows the seven groups we used during the analysis. These groups capture how rejections look at the level of specific tools or actions, before we merge them into broader categories of causes.

For each group, human annotators look at 100 randomly selected examples. They review the full context, including the serialized state, the suggestion made by the policy, and the action the operator chose instead. Annotators write free-form notes explaining why the rejection happened and assign

each example an error category and a more specific subcategory. For instance, an annotator might note that the model action is actually acceptable, with the subcategory being that the operator simply added softening language or emojis to the message. We then combine all observed categories into a single unified taxonomy and estimate how common each category is by scaling the proportions from the annotated samples to the full set of rejections collected over four days. It is worth noting that the subcategory proportions are approximate, since they come from samples of 100 examples per group and are then projected onto the entire dataset. Figure 2 shows the estimated distribution of rejection reasons across four high-level categories.

E.2 Typical categories and failure modes.

We further summarize the most common patterns observed within the four high-level categories.

Acceptable but rejected (37.2%). These cases reflect divergence between the model’s suggestion and operator preference or workflow style rather than a strictly incorrect action:

- **Minor text edits:** the operator rewrites the suggested message to be more polite, softer, shorter, or to add small clarifying details.
- **Optional verification steps:** the operator performs a non-mandatory check (e.g., opening an additional tab/panel) before sending a message or closing.
- **Valid alternative workflow variants:** the operator follows a different but permissible procedure variant (e.g., performs an intermediate step before a standard action).
- **Preference-driven handling of edge cases:** e.g., foreign-language chats where operators prefer to rewrite the message manually even if the model output is reasonable.

Implication: A simple binary choice between accept and reject mixes up personal preference with actual correctness. A practical next step would be to collect more detailed, graded feedback. For example, annotators could choose from options like acceptable as-is, acceptable with minor edits, incorrect, missing verification, or wrong workflow route. This would help separate the training signals that are truly useful from those that just reflect individual style.

Bucket (tool-oriented)	#	%
C1: model suggests a templated/text action, operator performs a different BPM action	2065	22.0
C2: model suggests free-form text, operator performs a different non-text action	1879	20.1
C3: model suggests closing, operator continues the workflow	1899	20.3
C4: model and operator both send text but content differs	1792	19.1
C5: operator edits/softens the suggested text	766	8.2
C6: operator performs a different UI action/control within a procedure	606	6.5
C7: model suggests a UI click, operator writes in chat instead	363	3.9
Total	9370	100.0

Table 9: Intermediate tool-oriented grouping of rejected events (counts and percentages over 9370 rejections).

1142	Environment limitations (6.9%). Here the rejection is driven by missing capabilities or incomplete observability:	providing a gold screen plan primarily improves routing-related tools.	1176
1143			1177
1144			
1145			
1146	<ul style="list-style-type: none"> • Non-text modalities: customers provide key information via images attachments that are not available to a text-only model. 	<ul style="list-style-type: none"> • Missing required steps or premature termination: the policy suggests closing the conversation or sending a message when the correct resolution requires additional actions, or a chain of actions, to be completed first. 	1178
1147			1179
1148	<ul style="list-style-type: none"> • Unsupported actions: workflows requiring phone calls or call outcomes that the copilot is not allowed or able to execute. 	<ul style="list-style-type: none"> • Conversation understanding failures: ignoring the latest user question, answering only one of multiple questions, or producing a response that does not address the user intent. 	1180
1149			1181
1150			1182
1151	<ul style="list-style-type: none"> • Logging and UI observability gaps: in some cases, important data is missing or unclear due to issues with logging, such as certain forms not being tracked, forwarded messages not being captured, or bugs in the interface. This prevents the model from seeing the evidence it needs. 	<ul style="list-style-type: none"> • Text generation quality issues: repeated content, overly confident claims that are not supported by evidence, and time-sensitive phrasing mistakes such as inappropriate greeting behaviour. 	1183
1152			1184
1153			1185
1154			1186
1155			1187
1156			1188
1157			1189
1158	<ul style="list-style-type: none"> • External knowledge not present in the environment: operators sometimes use information that exists outside the platform, for example details about ongoing operational incidents, which is not shown in the interface or included in the context. 	<ul style="list-style-type: none"> • Context construction and truncation artifacts: manual inspection suggests that a significant share of errors may be caused by the way context is assembled and trimmed. For example, crucial messages or action history may be missing from the input. In our internal analysis, such artifacts could account for up to roughly 26% of rejection cases, which motivates systematic context logging, validation, and more robust trimming strategies. 	1190
1159			1191
1160			1192
1161			1193
1162			1194
1163			1195
1164	Implication: These cases define the boundary of safe automation and motivate explicit stop-lists, fallbacks, and future modality expansion.		1196
1165			1197
1166			1198
1167	Model error (43.2%). These are cases where the suggestion is incorrect given the available context and supported capabilities. The most frequent failure modes include:	<ul style="list-style-type: none"> • Template and standard-form mismatches: cases where the model should send, or should not send, a standard-form or template message, or where it should verify user or account state before sending. Internal estimates suggest that up to roughly 9% of rejections may be related to such template usage patterns. 	1199
1168			1200
1169			1201
1170			1202
1171	<ul style="list-style-type: none"> • Workflow routing and navigation errors: the policy navigates to the wrong screen or procedure, selects a wrong control, or fails to switch between relevant UI panels. This aligns with our oracle plan experiment, where 		1203
1172			1204
1173			1205
1174			1206
1175			1207
		Implication: The mix of errors motivates several directions. First, selective execution and per-screen	1208
			1209
			1210

1211 thresholds can help reduce long-horizon drift. Sec- 1261
1212 ond, planner-like context can reduce routing ambi- 1262
1213 guity. Third, separating tool actions from free-form 1263
1214 text generation can help address text-specific er- 1264
1215 rors. Continuous monitoring targeting these failure 1265
1216 modes should support all three directions. 1266

1217 **Other (12.7%).** This bucket contains heteroge- 1267
1218 neous, low-frequency patterns that do not form sta- 1268
1219 ble subcategories in the 4-day sample. We treat it 1269
1220 as a tail distribution and prioritize categories above 1270
1221 for targeted data collection and mitigation. 1271

1222 A key observation is that rejections do not al- 1272
1223 ways mean the suggestion was wrong. A substan- 1273
1224 tial share of them, 37.2%, corresponds to actions 1274
1225 that are acceptable but still rejected. This can hap- 1275
1226 pen because of operator preference, optional steps, 1276
1227 or minor edits. This finding highlights that binary 1277
1228 accept or reject feedback is inherently noisy in a 1278
1229 copilot setting and should be treated as an imper- 1279
1230 fect supervision signal. In practice, reducing this 1280
1231 noise requires deliberate operator training and on- 1281
1232 boarding. Operator groups need guidance on how 1282
1233 to work with copilot suggestions, when to accept 1283
1234 them, and how to handle minor adjustments. In ad- 1284
1235 dition, collecting richer feedback, such as graded la-
1236 bels or standardised rejection reason codes, would
1237 make the signal more consistent and actionable.
1238 This in turn would enable more reliable critic train-
1239 ing and more effective automation improvements.

1240 F Resolution Correctness Criteria

1241 This appendix presents the methodology used by 1241
1242 the Quality Department to evaluate employee per- 1242
1243 formance during customer interactions. Each in- 1243
1244 teraction is assigned a resolution correctness score 1244
1245 ranging from 0 to 100 points. 1245

1246 The daily sample size was chosen to support 95% 1246
1247 confidence with approximately ± 3 points precision 1247
1248 assuming $\sigma = 30$. While 400 chats are sampled per 1248
1249 slice per day, statistical power improves as samples 1249
1250 accumulate over time. Quality was assessed using 1250
1251 the Quality Department’s standard employee eval- 1251
1252 uation rubric for both the baseline workflow and 1252
1253 the copilot-enabled workflow, including selective 1253
1254 automation sessions handled fully by the copilot. 1254
1255 This is possible because the copilot is trained via 1255
1256 behavior cloning to follow the same operating pro- 1256
1257 cedures as human agents, so no copilot-specific 1257
1258 scoring form is required. Each day, 400 chats were 1258
1259 randomly sampled per slice scored on a 0–100 res- 1259
1260 olution correctness scale by four trained evaluators. 1260

1261 Inter-rater agreement was monitored using a 3-way 1261
1262 overlap design on a subset of chats. Twelve trained 1262
1263 evaluators annotated 100 chats blindly each per day, 1263
1264 producing 400 unique chats per day with each chat 1264
1265 independently scored by three evaluators. We quan- 1265
1266 tified consistency of the final 0–100 quality score 1266
1267 using the intraclass correlation coefficient (ICC; 1267
1268 absolute agreement), which indicated substantial 1268
1269 agreement (ICC ≈ 0.8). 1269

1270 To reduce automation bias, the same standard- 1270
1271 ized rubric and identical scoring criteria were ap- 1271
1272 plied to all interactions regardless of assistance 1272
1273 condition. The evaluation is conducted using a 1273
1274 structured form consisting of several blocks, which 1274
1275 are listed in Table 10. Evaluators select a specific 1275
1276 issue corresponding to each observed criterion in 1276
1277 each block. Depending on the severity of that is- 1277
1278 sue, points may be deducted from the total score. 1278
1279 The first block includes general criteria that apply 1279
1280 to the communication as a whole. The remaining 1280
1281 blocks describe specific types of issues that may 1281
1282 occur within the interaction. When a specific issue 1282
1283 is identified, points are deducted from the overall 1283
1284 score in proportion to its severity. 1284

Block	Criterion	Options / Issues
General	Whether the Customer's Issue was Resolved	Issue resolved Issue not resolved Provided recommendations for resolution Impossible to resolve System limitations
		Understanding of the Customer's Question or Need
Customer Issue Resolution	Correctness of Consultation	Provided misinformation, critical consequences Provided misinformation, no critical consequences Failed to communicate information, critical consequences Failed to communicate information, no critical consequences Performed an action not requested by the customer
	Interaction with the system	Entered incorrect data Failed to open the required script Closed the chat out of time Other system error, critical consequences Other system error, no critical consequences Failed to complete identity verification Incorrect transfer Failed to escalate a case, critical consequences Failed to escalate a case, no critical consequences Incorrect session termination or disconnection Did not offer unified customer proposition (UCP) Incorrectly marked UCP status Did not handle UCP-related objections
	Resolution Speed	Provided irrelevant information Kept the customer waiting excessively
	Escalation (Requesting Assistance)	Failed to contact the support line, critical consequences Failed to contact the support line, no critical consequences Did not attempt to resolve the issue independently
	Vocabulary and Word Choice	Filler words Overly templated or dry phrases Did not greet the customer Improper or impolite form of address Overly long or rambling speech
Soft Skills	Communication Manner and Tone	Interrupted the customer fewer than 2 times Interrupted the customer more than 2 times Engaged in argument Unacceptable tone Inappropriate or unpleasant phrasing Dull or disengaged voice Audible sighs
	Clarity of Explanation	Not clear, critical consequences Not clear, no critical consequences
	Grammatical Accuracy and Fluency	More than two grammatical errors Fewer than two grammatical errors
	Handling of Negative Customer Emotions	Made no attempt to smooth out negativity Provoked additional negativity Did not apologize No empathy or acknowledgment
	Other	Discussed sensitive topics Damaged brand image Self-identification error Looping, repeated words or messages

Table 10: Quality assessment form with detailed options for each criterion.

GENERAL INSTRUCTIONS:
You are assisting an operator of a bank's support chat. Operator will control all your steps so
↪ listen to
all messages from "system" role and clearly describe the reasoning behind your actions. The
↪ following messages from the "system"
role do not override these instructions, but only indicate how to proceed with a particular step.
You'll get messages from bank clients and you must use tool calling to solve their problems.
You must hold a conversation in Russian language.

Use information about client to call a proper tool with parameters

PROCEDURES:
Your most common task will be interacting with procedures. Procedures are our internal tool to
↪ solve client problems. They contain information for you,
messages for client, or even scripts to manage client products. Use tool to interact with
↪ procedures that are displayed on operators screen.
Your called tools will be performed on operators computer.

PROCEDURE INTERACTION RULES:

1. Search procedure as first interaction;
2. When call search procedure use one most common word.
3. Remember that its cheaper for us if you do action in procedure rather than you write message to
↪ client;
4. Don't communicate with client until you interact with procedure;
5. If there is a form to fill on the procedure screen fill it accordingly with respect to the
↪ information from the client;
6. Write message only after you have interacted with the procedure and have the solution for the
↪ clients problem;
7. Don't write message to client if there is actions to do in procedure, while procedure is still
↪ in progress;
8. Don't close chat until you solve clients problem. Closing with timer is more common than close
↪ without;
9. Don't wait for clients response until you wrote him a message;

Figure 4: Prompt template used for the prompting baseline.